**MA CODES**

**1** **HELLO WORLD WITHOUT MACRO**

section .data

msg db " Hello World "

msg\_len equ $-msg

section .text

global \_start

\_start:

mov rax,1

mov rdi,1

mov rsi,msg

mov rdx,msg\_len

syscall

mov rax,60

mov rdi,0

syscall

**HELLO WORLD USING MACRO**

section .data

msg db " Hello World : ", 0ah

msg\_len equ $-msg

section .bss

buffer resb 20

string equ $-buffer

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

print msg, msg\_len

read buffer, string

exit

**;Write X86/64 Basic ALP for understanding Macro, syscall , data and text section.**

section .data

msg db "Enter name: ", 0ah

msg\_len equ $-msg

msg1 db "Name: ", 10

msg1\_len equ $-msg1

section .bss

buffer resb 20

string equ $-buffer

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

print msg, msg\_len

read buffer, string

print msg1, msg1\_len

print buffer, string

exit

**Write X86/64 ALP to accept a string and to display its length**

section .data

msg1 db"Enter the string:",10

msg1\_len equ $-msg1

msg2 db"Entered string is: "

msg2\_len equ $-msg2

msg3 db"The length of string is: "

msg3\_len equ $-msg3

msg4 db" ",10

msg4\_len equ $-msg4

section .bss

buffer resb 20

buffer\_len equ $-buffer

char\_ans resb 2

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

print msg1, msg1\_len

read buffer, buffer\_len

call disp\_proc

;dec rax

print msg2, msg2\_len

print buffer, buffer\_len

exit

disp\_proc:

mov rbx, 16

mov rcx, 2

mov rsi, char\_ans+1

cnt:

mov rdx, 0

div rbx

cmp dl, 09h

jbe add30

add dl, 07h

add30:

add dl, 30h

mov [rsi], dl

dec rsi

dec rcx

jnz cnt

print msg3, msg3\_len

print char\_ans, 2

print msg4, msg4\_len

ret

**Write an X86/64 ALP to count number of positive and negative numbers from the array.**

section .data

arr64 dq -111H, 222H, 333H, -444H, 777H, 666H

n equ 6

pmsg db 10,"The no. of positive elements from 64\_bit array : "

pmsg\_len equ $-pmsg

nmsg db 10,"The no. of negative elements from 64\_bit array : "

nmsg\_len equ $-nmsg

section .bss

p\_cnt resq 1

n\_cnt resq 1

char\_ans resb 02

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

mov rsi, arr64

mov rcx, n

mov rbx, 0

mov rdx, 0

next\_num:

mov rax, [rsi]

shl rax, 1

jc negative

positive:

inc rbx

jmp next

negative:

inc rdx

next:

add rsi, 8

dec rcx

jnz next\_num

mov [p\_cnt], rbx

mov [n\_cnt], rdx

print pmsg, pmsg\_len

mov rax, [p\_cnt]

call disp\_proc

print nmsg, nmsg\_len

mov rax, [n\_cnt]

call disp\_proc

exit

disp\_proc:

mov rbx, 16

mov rcx, 2

mov rsi, char\_ans+1

cnt:

mov rdx, 0

div rbx

cmp dl, 09h

jbe add30

add dl, 07h

add30:

add dl, 30h

mov [rsi], dl

dec rsi

dec rcx

jnz cnt

print char\_ans, 2

ret

**Write X86/64 ALP to convert 4-digit Hex number into its equivalent BCD number.**

section .data

hmsg db 10,"Enter 4 digit HEX number : "

hmsg\_len equ $-hmsg

ebmsg db 10,"Equivalent BCD number : "

ebmsg\_len equ $-ebmsg

ermsg db 10,"Enter valid HEX number !!" ,10

ermsg\_len equ $-ermsg

section .bss

buf resb 5

char\_ans resb 1

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

call Hex\_BCD

exit

Hex\_BCD:

print hmsg, hmsg\_len

call accept

mov ax, bx

mov bx, 10

xor bp, bp

back:

xor dx, dx

div bx

push dx

inc bp

cmp ax, 0

jne back

print ebmsg, ebmsg\_len

back1:

pop dx

add dl, 30h

mov[char\_ans], dl

print char\_ans, 1

dec bp

jnz back1

accept:

read buf, 5

mov rcx, 4

mov rsi, buf

xor bx, bx

next\_byte:

shl bx,4

mov al, [rsi]

cmp al,'0'

jb error

cmp al, '9'

jbe sub30

cmp al, 'A'

jb error

cmp al, 'F'

jbe sub37

cmp al, 'a'

jb error

cmp al, 'f'

jbe sub57

sub57:

sub al,20h

sub37:

sub al,07h

sub30:

sub al,30h

add bx, ax

inc rsi

dec rcx

jnz next\_byte

ret

error:

print ermsg, ermsg\_len

exit

**Write X86/64 ALP to convert 5- digit BCD number into its equivalent HEX number.**

section .data

bmsg db 10,"Enter 5 digit BCD number : "

bmsg\_len equ $-bmsg

ehmsg db 10,"Equivalent HEX number : "

ehmsg\_len equ $-ehmsg

section .bss

buff resb 6

char\_ans resb 4

ans resw 1

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

call BCD\_HEX

exit

BCD\_HEX:

print bmsg, bmsg\_len

read buff, 6

mov rsi, buff

xor rax, rax

mov rbp, 5

mov rbx, 10

next:

xor cx, cx

mul bx

mov cl, [rsi]

sub cl, 30H

add ax, cx

inc rsi

dec rbp

jnz next

mov [ans], ax

print ehmsg, ehmsg\_len

mov ax, [ans]

call disp\_proc

ret

disp\_proc:

mov rbx, 16

mov rcx, 4

mov rsi, char\_ans+1

cnt:

mov rdx, 0

div rbx

cmp dl, 09h

jbe add30

add dl, 07h

add30:

add dl, 30h

mov [rsi], dl

dec rsi

dec rcx

jnz cnt

print char\_ans, 4

ret

**Write X86/64 ALP to perform non-overlapped block transfer without string specific instructions.**

section .data

sblock db 10h, 20h, 30h, 40h, 50h, 60h, 70h

dblock db 0h, 0h, 0h, 0h, 0h, 0h, 0h ;dblock times 7 db 0

smsg db 10,"Source Block is : "

smsg\_len equ $-smsg

dmsg db 10,"Destination Block is : "

dmsg\_len equ $-dmsg

space db " "

section .bss

char\_ans resb 2

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

print smsg, smsg\_len

mov rsi, sblock

call block\_display

print dmsg, dmsg\_len

mov rsi, dblock

call block\_display

call block\_transfer

print smsg, smsg\_len

mov rsi, sblock

call block\_display

print dmsg, dmsg\_len

mov rsi, dblock

call block\_display

exit

block\_transfer:

mov rsi, sblock

mov rdi, dblock

mov rcx, 7

back:

mov al, [rsi]

mov [rdi], al

inc rsi

inc rdi

dec rcx

jnz back

;with string

;cld ;inc rsi & rdi

;rep movsb

ret

block\_display:

mov rbp, 7

next\_num:

mov al, [rsi]

push rsi

call disp\_proc

print space, 1

pop rsi

inc rsi

dec rbp

jnz next\_num

ret

disp\_proc:

mov rbx, 16

mov rcx, 2

mov rsi, char\_ans+1

cnt:

mov rdx, 0

div rbx

cmp dl, 09h

jbe add30

add dl, 07h

add30:

add dl, 30h

mov [rsi], dl

dec rsi

dec rcx

jnz cnt

print char\_ans, 2

ret

**;Write X86/64 ALP to perform overlapped block transfer with string specific instructions.**

section .data

sblock db 11h, 12h, 13h, 14h, 15h

dblock db 0h, 0h, 0h, 0h, 0h ;dblock times 5 db 0

smsg db 10,"Source Block is : "

smsg\_len equ $-smsg

dmsg db 10,"Destination Block is : "

dmsg\_len equ $-dmsg

space db " "

section .bss

char\_ans resb 2

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

print smsg, smsg\_len

mov rsi, sblock

call block\_display

print dmsg, dmsg\_len

mov rsi, dblock-2

call block\_display

call block\_transfer

print smsg, smsg\_len

mov rsi, sblock

call block\_display

print dmsg, dmsg\_len

mov rsi, dblock-2

call block\_display

exit

block\_transfer:

mov rsi, sblock+4

mov rdi, dblock+2

mov rcx, 5

back:

mov al, [rsi]

mov [rdi], al

dec rsi

dec rdi

dec rcx

jnz back

;with string

;std ;dec rsi & rdi

;rep movsb

ret

block\_display:

mov rbp, 5

next\_num:

mov al, [rsi]

push rsi

call disp\_proc

print space, 1

pop rsi

inc rsi

dec rbp

jnz next\_num

ret

disp\_proc:

mov rbx, 16

mov rcx, 2

mov rsi, char\_ans+1

cnt:

mov rdx, 0

div rbx

cmp dl, 09h

jbe add30

add dl, 07h

add30:

add dl, 30h

mov [rsi], dl

dec rsi

dec rcx

jnz cnt

print char\_ans, 2

ret

**;Write X86/64 ALP to detect protected mode and display the values of GDTR, LDTR, IDTR, TR and MSW Registers.**

section .data

msg1 db "Processor is in protected mode",10

msg1\_len equ $-msg1

msg2 db "Processor is not in protected mode",10

msg2\_len equ $-msg2

mgdt db "Value of GDTR: "

mgdt\_len equ $-mgdt

mline db " ",10

mline\_len equ $-mline

mldt db "Value of LDTR: "

mldt\_len equ $-mldt

midt db "Value of IDTR: "

midt\_len equ $-midt

mmsw db "Value of MSW: "

mmsw\_len equ $-mmsw

section .bss

LDTR resw 1

GDTR resw 3

MSW resw 1

IDTR resw 3

TR resw 1

char\_ans resb 4

%macro print 2

mov rax, 1

mov rdi, 1

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro read 2

mov rax, 0

mov rdi, 0

mov rsi, %1

mov rdx, %2

syscall

%endmacro

%macro exit 0

mov rax, 60

mov rdi, 0

syscall

%endmacro

section .text

global \_start

\_start:

SMSW [MSW]

mov ax,[MSW]

shr ax,1

jc pmode

print msg2,msg2\_len

pmode:

print msg1,msg1\_len

jmp next

next:

SGDT [GDTR]

SLDT [LDTR]

STR [TR]

SIDT [IDTR]

SMSW [MSW]

print mgdt,mgdt\_len

mov ax,[GDTR+4]

call display

mov ax,[GDTR+2]

call display

mov ax,[GDTR]

call display

print mline,mline\_len

print mldt,mldt\_len

mov ax,[LDTR]

call display

print mline,mline\_len

print mmsw,mmsw\_len

mov ax,[MSW]

call display

exit

display:

mov rbx, 16

mov rcx, 2

mov rsi, char\_ans+1

cnt:

mov rdx, 0

div rbx

cmp dl, 09h

jbe add30

add dl, 07h

add30:

add dl, 30h

mov [rsi], dl

dec rsi

dec rcx

jnz cnt

print char\_ans, 2

ret